

A Performance Analysis of the BigBangwidth Lightpath Accelerator

Eric Weigle

eweigle@ucsd.edu

Concurrent Systems Architecture Group (CSAG)

University of California – San Diego

La Jolla, CA 92093-0114

Abstract

We study the performance characteristics of BigBangwidth’s Lightpath Accelerator 306 in a high performance cluster, and discuss the expected performance in a grid environment.

Keywords: BigBangwidth, Packet switching, Circuit switching, Grids

1. Introduction

Packet-switched and circuit-switched networks each have positive and negative characteristics. Packet-switched networks work well for short, unpredictable transfers and have low configuration overhead, but generally do not provide good quality of service. Circuit switched networks provide better service and efficiency for longer or more predictable transfers, but at the cost of a higher configuration overhead and wasting some available bandwidth. BigBangwidth’s Lightpath Accelerator attempts to combine the two technologies in a sensible way to exploit the capabilities of both approaches.

The Lightpath Accelerator and associated software detect high-bandwidth flows over a packet-switched network and move them onto a separate circuit-switched network. By separating these bulk transfers, both they and the remaining traffic on the packet-switched network should experience better performance. This document evaluates the effectiveness of this technique.

1.1. Hardware and Software

Figure 1 shows the configuration of our hardware in all tests. The front-end and compute nodes are part of a Rocks cluster [4] based off of Red Hat Linux [3].

The thin lines in the figure are copper gigabit Ethernet while the thicker lines are single-mode fiber. The dotted line is a serial connection.

The front-end and compute nodes are ProMicro machines, they have SuperMicro SUPER X5DPA-GG motherboards, dual Intel Xeon 2.4GHz CPUs with hyperthreading disabled, 2x1GB of ECC DDR DIMM chips in dual-memory bank configuration, and dual Intel 82541 Gigabit Ethernet controllers on-board using the Intel e1000 driver.

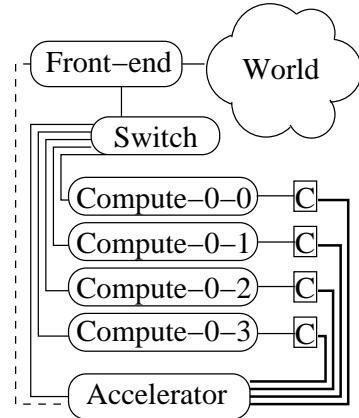


Figure 1. Hardware used in the experiments

The switch is a 24-port gigabit Ethernet (copper) Dell Powerconnect 5224. The “C” boxes are converters from copper to SC single-mode fiber made by Omnitron Systems Technology; the Flexpoint GX.

The Accelerator is a BigBangwidth Lightpath Accelerator model 306D, running software version 1.5.1/build 309. It is a six-port all-optical (SC single-mode fiber) circuit switching device. It has two management interfaces: a copper 10Mb Ethernet interface and a serial interface.

2. Experiments

We perform a simple set of tests to determine the network performance characteristics for this hardware. In particular we are interested in the bandwidth and latency available over the packet switched network, over the circuit switched network, and the latency to create/destroy circuits (begin and end offloading).

We use the standard Linux `ping` program to test latency and NLANR's `iperf` version 1.7.0 [2] to test bandwidth. As discussed above, one interface is attached to the switch on a cluster-private ($10.*.*$) network, while the other is dedicated to the accelerator ($192.168.*.*$) circuit-switched network even if not being used in any circuit at a given time. All nonlocal resource access is performed by way of the front-end node.

3. Results

We begin in Section 3.1 by looking at the bandwidth over the packet-switched network. The test setup is the expected configuration for an average cluster connected via gigabit Ethernet so provides a useful baseline for comparison with the circuit-switched performance. Section 3.2 presents the latency results. A summary of all results are given in Table 1.

3.1. Bandwidth Results

We use `iperf` with everything set at the defaults—10 second unidirectional transfers, 16KB TCP windows, 1500 byte Ethernet MTU. For these short flows we see performance ranging between 900-940Mbps. This is the payload data rate—accounting for packet headers (14 bytes Ethernet, 20 bytes IP, 32 bytes TCP) the actual line data rate is a few percent higher and near the theoretical maximum. Longer flows (60 seconds or more) will achieve sustained rates of about 940Mbps.

The CPU load in these transfers is low due to the use of DMA and implicit interrupt coalescing via the NAPI in the e1000 driver. This allows multiple packets to be copied from the NIC at once in a polling fashion rather than forcing the context-switch and copy overhead on every packet. Testing `iperf` over the loopback interface on a single machine achieves 2.34Gbps for a 60-second transfer. This means that the machine can handle all the send and receive requests at that rate, and implies (ignoring the differences between the loopback and e1000 driver) that when we are only handling sends or only handling receives we should never be CPU or memory limited in any of our unidirectional tests, even when using both NICs concurrently. The client

and server `iperf` processes report each using no more than 12% of the CPU in this case (via `top`).

Bidirectional bandwidth tests show a very different picture. Again using `iperf` with everything at default, we see just over 400Mbps each send and receive performance for a single flow. The aggregate rate is never more than 850Mbps. This suggests that the driver is effectively limited to half-duplex mode, even though both it and the switch claim full duplex. Some tuning, such as increasing the size of the NIC's send/receive queues, would hopefully improve the performance.

For the circuit-switched network we duplicated the tests done earlier. Circuits were statically pre-configured between ports on the accelerator by hand, by logging in over the serial interface and using the command-line interface. The `pipecreator` command in newer versions of the software could be used for the same purpose. In general the performance through the accelerator is the same as that through the packet-switched network. This is unsurprising, because of three factors: our switch is nonblocking, the copper to SMF converters are merely changing on-the-wire formats, and the accelerator does not touch the packets.

We also tested two simultaneous connections going through both the switch and the accelerator concurrently to determine whether the flows could interfere with each other. There were no detectable changes in the performance of concurrent flows as compared to single flows—different streams are effectively independent in this LAN environment. Similarly, increasing the TCP window sizes up to 256KB (far more than the worst-case bandwidth \times delay product) does not change the results for poorly-performing flows.

Unfortunately, the prior evaluation is not particularly realistic. The expected use of the accelerator is in a wide-area network; a packet-switched WAN would have congestion and variable delay, while a circuit-switched WAN would have no congestion and much more consistent delay. In such a case, we would expect single-flow performance over the accelerator to be much better than over the packet-switched network (primarily by avoiding packet loss in routers), and multiple flow performance to also be superior by guaranteeing stream independence.

Testing this in a laboratory using WAN emulators would give useless results, reflecting only the assumptions on loss over the packet switched-network and the performance of TCP with high-delay. Although we expect everything discussed here to correspond well to the real world, a real network with live traffic is necessary to fully verify the system. We intend to repeat these experiments once real-world OptIPuter resources become available in the wide area.

	Packet-Switched		Circuit-Switched	
	Min	Peak	Min	Peak
loopback-60s (Gbps)	2.30	2.34	n/a	n/a
iperf-10s (Mbps)	891	940	904	941
iperf-60s (Mbps)	913	940	931	939
2-dir iperf (Mbps)	754	845	710	896

Table 1. Bandwidth Performance Summary

The results discussed above were obtained with hand-configured connections and did not take advantage of the pfd/epd (Pipe Fitting Daemon/End Point Daemon) pair which automatically offload large flows from the packet-switched network onto the circuit-switched network. We now consider two unidirectional flows coming from one node to two other nodes, and see how the system performs. We run these iperf tests 60 seconds and observe the performance of both flows. Table 2 shows the results.

Two flows over the packet-switched network perform much like a single flow performs, and are limited by the maximum payload data rate of the single interface (about 956Mbps). When we use the second NIC to form a circuit between the sender and one of the destination hosts, we double the available bandwidth and expect to see improved performance. Flow 1 is started slightly before flow 2, and achieves marginally higher performance because of it.

When we pre-configure a circuit by hand (for flow 1) we see the best performance, because the two flows never go across the same NICs. This configuration is somewhat irritating and is obviously quite visible to the user, so there is a tradeoff. It is unclear why the bandwidth achieved by the flow through the accelerator is higher than that of the other flow; possible reasons include reduced jitter or avoiding hardware level source-quench flow control.

When we allow the end-point daemon to detect the high bandwidth flows, it will create a circuit and move traffic without user intervention. It achieves slightly lower performance than in the static case because for the first several seconds of the connections, both are sharing the packet-switched interface before Flow 1 is moved to the newly created circuit. This transition is invisible to the user. When the flows terminate the connection is also automatically torn down.

3.2. Latency Results

The ping command was used to observe the round-trip time between various nodes over the two networks.

Experiment	Flow 1	Flow 2	Total
No circuits	474	472	946
Static circuits	730	698	1,428
Dynamic circuits	734	626	1,360

Table 2. Performance of Offload Daemons (Mbps)

Our results do not show the expected consistently low (50 microsecond) times. Instead they show great variability and cyclic behavior.

We begin with a discussion of performance over the packet-switched network, looking at the results of ping's once-a-second ping ICMP ECHO_REQUEST and ECHO_REPLY. Over the course of one five minute test, we see (in microseconds) a minimum RTT of 71, maximum RTT of 302, average of 186, and unexpectedly high standard deviation of 50. A graph of this data is much more enlightening.

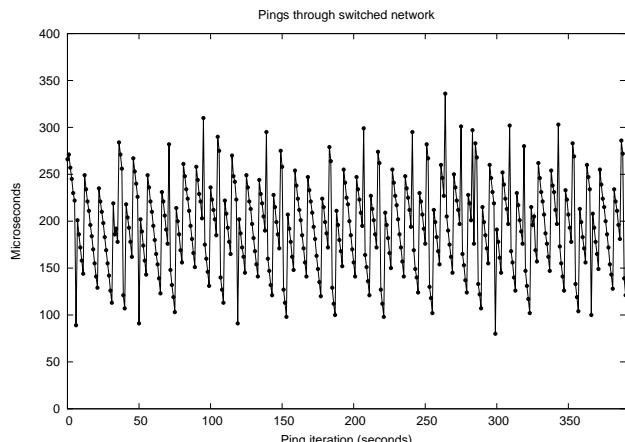


Figure 2. Ping time variability in packet-switched network

We see that there is an obvious cyclic behavior in the data. Figure 3 shows the same graph over the circuit switched network. We see essentially the same pattern, statistically within about 5% for all cases. The diagonal lines show that the strong modes within the data (more obvious here than in the switched data, but they are present there as well).

Together these two graphs show that the problems are a host/NIC issue rather than a network issue. Upon further thought we are led inevitably back to the e1000 driver and its NAPI implementation. The times in-

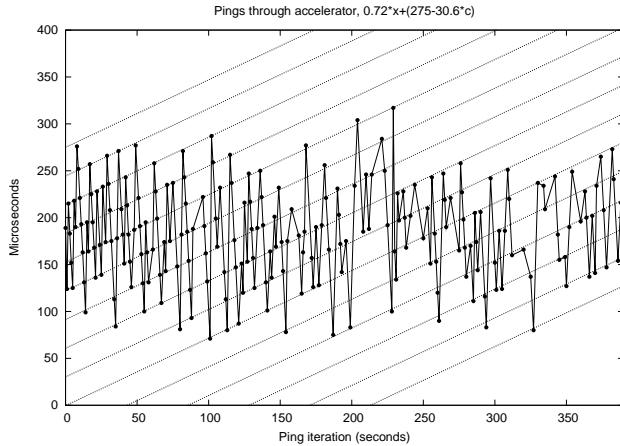


Figure 3. Ping time variability in circuit-switched network

volved are short enough that even the maximum is still far less than the scheduling quantum, 10ms. It uses a heuristic to do interrupt coalescing; depending on exactly when in the NIC polling cycle we make a request (once every $1+RTT+\epsilon$ seconds) we see a slightly different delay.

We also performed several 2000-packet ping floods, trying to get statistical performance information. These tests never lost a packet and showed some horrible results: in microseconds, a minimum RTT of 63, a maximum of 1079, and an average of 549. Cutting that in half to find the one-way delay gives an acceptable best-case of 31.5 microseconds, but an unacceptably large average of over a quarter of a millisecond. Over the circuit-switched network we see similar minimum and maximum values, but have an average about 80-90 microseconds higher.

Part of this is likely the kludgy way in which we are converting copper gigabit Ethernet to single-mode fiber and back again. We are also likely inducing short-term congestion due to high packet rates and causing the ‘pause’ based flow control mechanisms to activate at the data link level. This should probably be looked at in more detail.

One other measure of latency is the time to set up and tear down connections. When configuring by hand via the command-line interface it takes about 30 seconds; you have to log in to the machine, start up a serial connection, log in to the accelerator, tell it to create a connection (this command takes 1-2 seconds to return), then log out of everything. When the endpoint daemon does this automatically, it takes about six seconds; four seconds to detect the high bandwidth

flow, then two seconds to create the circuit and move new traffic on to it. Thus, flows need a duration of at least tens of seconds for the circuit switching to be useful. One nice aspect of the offloading is that there is no obvious “hiccup” as traffic is moved; everything functions normally invisibly to the user, and no packets are unduly delayed.

Over a real packet switched network with congestion and even more variable delay, when we change to the circuit the “hiccup” would be more noticeable; this is likely a good thing as it is supposed to improve performance. Explicitly characterizing the behavior during the switch-over phase would be useful, there is likely a lot of work going on ‘under the covers’. In particular it may be inducing another packet copy— this would be noticeable at higher data rates (which this hardware can not accommodate).

The driver has various parameters to tune the maximum interrupt delay, number of descriptors, etc. That the defaults produce such behavior shows that the only performance metric most people are concerned with is bandwidth. Note that we have not tuned anything, minimal tuning or use of an automatically tuned algorithm (gridFTP/drs/etc) may change the results.

3.3. Discussion

One question that arises is the overhead of the endpoint daemon flow tracking— does it affect bandwidth or latency? We ran tests with and without `epd` running on the two hosts in a connection to see if we could detect any performance impact. Our results did not reveal any such impact— while there must be some effect, it is well within the normal variability of the data, and hence is not detectable.

It is unclear whether the Lightpath Accelerator is solving the right problem in a grid environment. On the one hand, we require high bandwidth, low latency, low jitter connections (which the accelerator provides). On the other hand, we do not really want to keep half the network capacity of a machine idle most of the time. The performance impact cannot be truly known until we conduct real world experiments on live networks later in 2004. In particular, the wide-area network’s behavior may dramatically reduce the change the packet-switched performance. Other models, such as having an accelerator-like device connected to a switch in the network and offloading switch-to switch, or doing traffic shaping (bandwidth reservation) on a single shared NIC may be better for grid environments.

4. Conclusion

The device works as advertised, giving good performance in all test cases. Unfortunately, all performance results can be duplicated using solely the two NICs and a packet-switched network. The true ‘win’ with the device will be over the real world WAN when loss and variable bandwidth become issues. Future tests in this environment are required but we expect it will perform very well.

With respect to the new ProMicro machines, the packet latency patterns mean this hardware is not ideal for scientific computing, and the performance is much less than we would like it to be. This is most likely because the interfaces are integrated onto the motherboard and are sharing resources, rather than being full-fledged devices with their own memory/cpu/etc.

References

- [1] S. Lomas. Breaking the limits of packet switching. Big-Bangwidth Incorporated, March 2003.
- [2] NLANR Distributed Applications Support Team. Iperf TCP/UDP bandwidth measurement tool. <http://dast.nlanr.net/Projects/Iperf/>.
- [3] Red Hat Inc. and The Free Software Community. Red hat linux, 1994. <http://www.redhat.com/>.
- [4] SDSC Cluster Development Group. Rocks cluster distribution. <http://www.rocksclusters.org>.

A. Accelerator Administration

This section gives brief information on how install and use the accelerator software.

First, you must log in to all the nodes which are connected to the accelerator and install the software. SSH works fine for this; the front-end node for the cluster is csag-226-21. Then install the packages on all the nodes. As root this simply requires an

```
rpm --install bbw-accel-1.5.1-309.i386.rpm
```

There is a ROCKS mechanism for doing this on all nodes simultaneously, but it’s easier to just ssh in for a single package.

Configuration of the packages is a pain. Config files are XML, and live in /usr/bbw. Each node must be configured independently with its own hostname and IP and the package isn’t smart enough to take this from the Ethernet interface. You must configure each tool independently as well, the pipe-fitter daemon (pfd) versus the endpoint daemon (epd) versus the manager tool (itmanager) versus the topology of the network. Lots of the information is duplicated and must be kept consistent. A GUI for this would be very nice to have.

If you get something wrong, it may sort of work but then fail. You’ll have to check the log files to figure out what’s going on. I suggest you build off of the config file versions I’ve created, which I have saved and can give you.

To log in via the serial console you can use minicom on the front-end node. I’ve created a config file with the appropriate settings:

```
minicom le-accel
```

Hit enter and it should bring up a login prompt. Use admin/admin to log in. Type help. This should give the version (1.5.1 build 309 or better) and a list of commands:

- connection <port1> <port2>
- no connection [<port1> <port2>]
- default connection
- [no] connection cycle
- show connections
- clear connections
- [show] help
- ip address [dhcp — <addr>]
- no ip address dhcp
- [no — show] ip address

- ip default-gateway <ip-address>
- [no — show — default] ip default-gateway
- [show — default] ip netmask
- show ip interface
- [show] version
- quit

All except for the connection[s] commands are just to configure the device the first time. You can obviously ask for help on any individual command. Here's all you probably want to do however:

Make a connection by hand between ports 1 and 4:

```
connection 1 4
```

Turn that connection off and make a new one 2–4

```
no connection 1 4
connection 4 2
```

Turn off all connections

```
clear connections
```

To do everything automatically you need:

1. Accelerator up with an IP, connected to switch or front-end
2. Properly configured pfd and topology on frontend
3. Properly configured epd and topology on compute nodes
4. Second interface on compute nodes up on the private private (not cluster-private) subnet

Then the epd will detect high flows, contact the pfd on the frontend, which will tell the switch to make a circuit, which will reply to the frontend, which will then let the compute nodes actually move their traffic over the connection. It's nice.

B. Gotchas

The accelerator doesn't seem to like subnet masks not equal to 24 bits. If you reboot it you may have to log in to it as user 'root' password 'bigbang' and use the Linux 'ifattach' command by hand.

If the accelerator gets an IP via DHCP and you later want to reconfigure it by hand, you **have** to use the ifattach command, this is a known bug in their firmware and will be fixed in a later version.